Report No. NAWCADWAR-94112-50

MODIFIED GAUSSIAN ELIMINATION FOR ADAPTIVE BEAM FORMING USING RNS ARITHMETIC

Barry J. Kirsch
Mission Avionics Technology Department (Code 5051)
NAVAL AIR WARFARE CENTER
AIRCRAFT DIVISION WARMINSTER
P.O. Box 5152
Warminster, PA 18974-0591

Peter R. Turner, Ph.D.
Mathematics Department
U.S. NAVAL ACADEMY
Annapolis, MD 21402

1 SEPTEMBER 1994

FINAL REPORT
Period Covering 6 July 1992 to 11 September 1992

Prepared for
OFFICE OF NAVAL RESEARCH
800 N. Quincy Street
Arlington, VA 22217

NOTICES

REPORT NUMBERING SYSTEM — The numbering of technical project reports issued by the Naval Air Development Center is arranged for specific identification purposes. Each number consists of the Center acronym, the calendar year in which the number was assigned, the sequence number of the report within the specific calendar year, and the official 2-digit correspondence code of the Command Officer or the Functional Department responsible for the report. For example: Report No. NADC-88020-60 indicates the twentieth Center report for the year 1988 and prepared by the Air Vehicle and Crew Systems Technology Deartment. The numerical codes are as follows:

CODE	OFFICE OR DEPARTMENT
00	Commander, Naval Air Development Center
01	Technical Director, Naval Air Development Center
05	Computer Department
10	AntiSubmarine Warfare Systems Department
20	Tactical Air Systems Department
30	Warfare Systems Analysis Department
40	Communication Navigation Technology Department
50	Mission Avionics Technology Department
60	Air Vehicle & Crew Systems Technology Department
70	Systems & Software Technology Department
80	Engineering Support Group
90	Test & Evaluation Group

Reviewed By:	[illegible] Branch Head	Date:	8/9/94
Reviewed By:	[illegible] Division Head	Date:	8/12/94
Approved By:	[illegible] Director/Deputy Director	Date:	10/28/94

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
1 September 1994

3. REPORT TYPE AND DATES COVERED
Final 6 Jul 92 - 11 Sep 92

4. TITLE AND SUBTITLE
MODIFIED GAUSSIAN ELIMINATION FOR ADAPTIVE BEAMFORMING USING RNS ARITHMETIC

5. FUNDING NUMBERS

6. AUTHOR(S)
Barry J. Kirsch
*Peter R. Turner, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Air Warfare Center
Aircraft Division Warminster
Code 5051
PO Box 5152
Warminster, PA 18974-0591

8. PERFORMING ORGANIZATION REPORT NUMBER
NAWCADWAR-94112-50

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
* Mathematics Department
US Naval Academy
Annapolis, MD 21402

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper is concerned with the development of a solution to the adaptive beamforming problem. The proposed solution consists of an algorithm-architecture-arithmetic combination which has the potential for very fast solution of the problem on a physically small platform which might be suitable for use on aircraft or sonobuoys. The arithmetic system used in the proposed solution is the RNS system implemented on an array of RNS processors which can be reassigned as the algorithm proceeds. The underlying algorithm is a modification of Gaussian elimination. The (non-RNS) division operations are eliminated in favor of some scaling and the adaptive use of the processor array to accommodate the growth in dynamic range which is implicit in this divisionless implementation.

14. SUBJECT TERMS

RNS arithmetic, Beamforming, Gauss elimination

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TABLE OF CONTENTS
REPORT NO. NAWC-XXXXX-XX

94112-50
NAWCADWAR

MODIFIED GAUSSIAN ELIMINATION FOR ADAPTIVE BEAMFORMING USING RNS ARITHMETIC

Barry J Kirsch
Naval Air Warfare Center
Code 5051
Warminster PA 18974

bkirsch@nadc.navy.mil

and

Peter R Turner
Mathematics Department
U S Naval Academy
Annapolis MD 21402

prt@usna.navy.mil

Abstract:

This paper is concerned with the development of a solution to the adaptive beamforming problem. The proposed solution consists of an algorithm-architecture-arithmetic combination which has the potential for very fast solution of the problem on a physically small platform which might be suitable for use on aircraft or sonobuoys. The arithmetic system used in the proposed solution is the RNS system implemented on an array of RNS processors which can be reassigned as the algorithm proceeds. The underlying algorithm is a modification of Gaussian elimination. The (non-RNS) division operations are eliminated in favor of some scaling and the adaptive use of the processor array to accommodate the growth in dynamic range which is implicit in this divisionless implementation.

Key Words:
RNS arithmetic, Beamforming, Gauss elimination

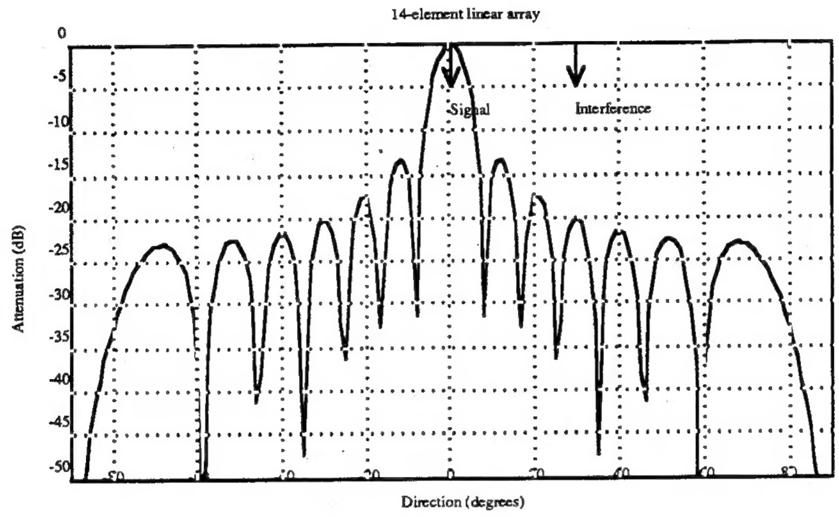
Accesion For		
NTIS CRA&I	[✓]	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

MODIFIED GAUSSIAN ELIMINATION FOR ADAPTIVE BEAMFORMING USING RNS ARITHMETIC

1. Introduction

A typical beam pattern produced by an uniformly weighted linear array of antenna or hydrophones is shown below.

FIGURE 1
Beam pattern for a uniformly weighted linear array



The horizontal axis is the physical angle of observation and the vertical axis represents the gain (attenuation) that the antenna array produces. If directional interference impinges the array, that interference will be attenuated by a given amount depending on the direction. We will have the most attenuation if the interferences falls in the direction of the nulls in the beam pattern. This is desireable but occurs only by chance. By choosing the appropriate amplitude and phase weighting of the antenna elements, we can steer the nulls in the direction of the interferences. The problem has been extensively studied and many solution algorithms have been developed. This paper describes one solution to this problem for a particular algorithm and suggests some novel processor implementations.

The particular concern of this paper is with obtaining solution *quickly* on a *physically small* processing unit for operation on platforms such as aircraft or sonobuoys. This requires that we seek nonstandard solution techniques. Clearly, speed of numerical processing is vital; this is the reason for choosing RNS arithmetic. Speed also dictates that nonstandard RNS operations be kept to an absolute minimum; this in turn places constraints on the algorithm. The algorithm-architecture combination proposed here is based on using Gauss elimination to solve the covariance matrix formulation of the problem.

The divisions implicit in the method are eliminated at the expense of requiring substantial growth in the dynamic range of the RNS system used. This growth is accommodated by a combination of the adaptive use of an array of RNS processors and some scaling to reduce the effect of the growth of the matrix elements.

In the remainder of this section, we describe the adaptive beamforming problem and some of the approaches that have been used for its solution.

In Section 2, we summarize briefly the relevant aspects of residue number systems, RNS arithmetic, and its extensions to complex RNS arithmetic. The architecture required for our solution is based on an array of RNS processors. The section finishes with a brief discussion of scaling in RNS arithmetic.

Section 3 is concerned with Gauss elimination and a description of the modifications to the standard algorithm which are required here. It is also in this section that the basic philosophy of the algorithm is presented. In Section 4, several of the subproblems and their associated difficulties are discussed along with proposed answers to these problems. The difficulties center on the questions of the growth of the matrix elements and the use of adaptive RNS-base extension and scaling to handle this growth. Scaling necessarily affects the accuracy of the solution; this effect is also discussed in this section.

Section 5 brings the ideas together in a detailed description of the overall elimination algorithm. In Section 6, the back substitution phase is described and, in Section 7, the modification to a Gauss-Jordan solution is considered.

1.1 Physical Problem and Basic Engineering Approach

A typical beamforming situation is shown in Figure 2. An array of N antenna elements are sampled at time k to form a complex snapshot vector $\vec{x}_k$. A collection of K of these snapshots constitute the $N \times K$ ($N<K$) data matrix X. Inner products between the data vector $\vec{x}_k$ and complex weights $\vec{w}$ form the complex scalar outputs y_k. For the time from 1 to K, the output vector $\vec{y} = \vec{w}^H X$. The problem is to determine the weights w_0, w_1, ...,w_{N-1} that will optimize the response $\vec{y}$ in some sense. When it is necessary to continually adjust the weights, we say that we are doing adaptive beamforming.
Thus we have

$$\text{Input} = \vec{x}(t) = \begin{pmatrix} x_0(t) \\ x_1(t) \\ \vdots \\ x_{N-1}(t) \end{pmatrix}$$

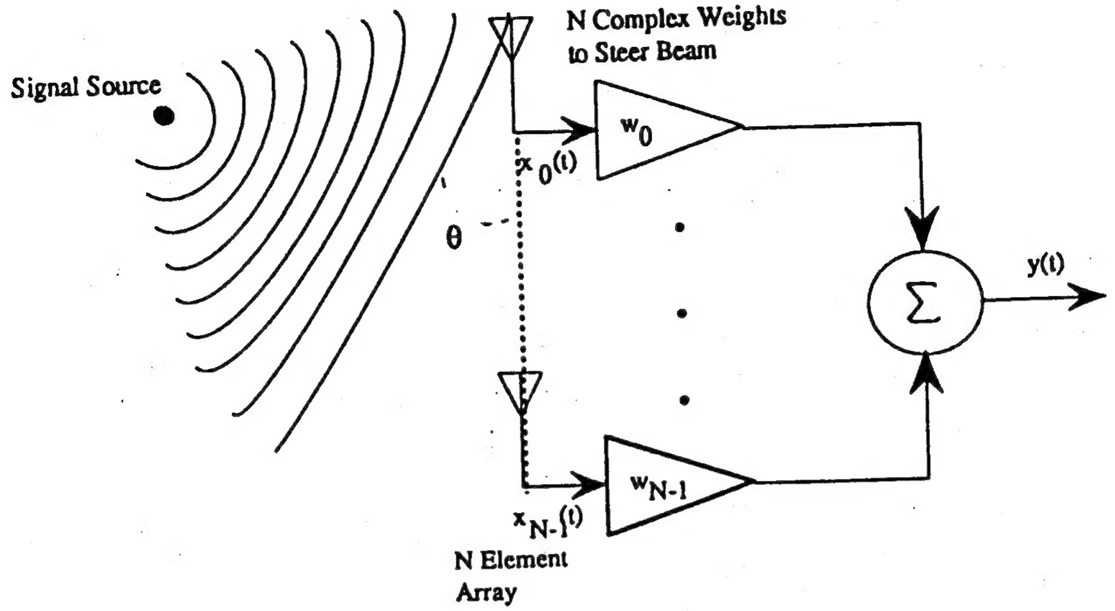
and seek

$$\text{Weights} = \vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{pmatrix}$$

The situation is illustrated in Figure 2 below.

There are various techniques used for solving the beamforming problem which fall into three basic categories.

FIGURE 2



1.2 Covariance Matrix methods of forming adapted weights

We can derive the optimal weights to minimize the mean-square error, MSE = $E[\epsilon^2]$, where the error signal, ϵ is the difference between the desired response and the output y.

$$\epsilon_k = d_k - y_k = d_k - \vec{w}^H \vec{x}_k$$
$$\epsilon_k^2 = d_k^2 - 2d_k \vec{w}^H \vec{x}_k + \vec{w}^H \vec{x}_k \vec{x}_k^H \vec{w}$$

Taking expected values of both sides yields

$$E[\epsilon^2] = \overline{\epsilon_k^2} = \overline{d_k^2} - 2\vec{w}^H \overline{\vec{x}_k d_k} + \vec{w}^H \overline{\vec{x}_k \vec{x}_k^H} \vec{w}$$

or

$$E[\epsilon^2] = \overline{d_k^2} - 2\vec{w}^H \vec{r}_{xd} + \vec{w}^H R_{xx} \vec{w} \tag{1.2.1}$$

To minimize this function, we set the gradient with respect to the weight vector equal to zero, that is,

$$\nabla\epsilon^2 = -2\vec{r}_{xd} + 2R_{xx}\vec{w} = 0 \rightarrow R_{xx}\vec{w} = r_{xd} \tag{1.2.2}$$

An approximation R to the correlation matrix R_{xx} (also called the covariance matrix for zero-mean data [8]) is formed from the $N \times K$ data matrix X. R_{xx} is the complex $N \times N$ matrix $R_{xx} = E[XX^H]$ which is an infinite time average. Since we only have a finite number K of snapshots, we use the estimated covariance matrix

$$\hat{R} = XX^H / K$$

The covariance matrix is always non-singular, and hence $\hat{R}$ is a positive definite Hermitian matrix, since statistically independent noise exists on the antenna elements. The noise correlation matrix is just $R_n = \sigma^2 I$, where σ^2 is the noise variance (power), and I is the identity matrix of size N. That is, the cross-correlation terms average out while the autocorrelation terms average to the variance of the noise. The data covariance matrix is made from the sum of the signal, jammer and noise covariance matrices: $R = R_s + R_j + R_n$.

The weight vector is found by solving the system $R\vec{w} = \vec{s}$ where *either*

(a) $\vec{s}$ could be the *steering vector* given by

$$\vec{s} = (1, e^{-j\phi}, e^{-2j\phi}, \ldots, e^{-(N-1)j\phi})^T$$

where $\phi = (2\pi d/\lambda)\sin\theta$ and θ is the desired look-angle with respect to the normal to the linear antenna array; d is the inter-element spacing and λ is the wavelength of the incoming signal at the carrier frequency, *or*

(b) $\vec{s}$ could be the *cross-correlation* vector

$$\vec{r}_{xd} = \mathrm{E}[\vec{x}_k d_k^*] \approx (X\vec{d}^H)/K$$

where d_k is the reference signal sampled at time k, $\vec{x}_k = (x_0, x_1, \ldots, x_{N-1})^T$ is the snapshot vector at time k, and, as before, $E[\bullet]$ is the expectation operator.

Covariance matrix algorithms which have been used for solving this problem include Gauss elimination, Cholesky decomposition, and the recursive least-squares (RLS) method based on the matrix inverse lemma [5, p.385].

1.3 Data matrix methods of forming adapted weights

The data matrix X is a complex $N \times K$ matrix, where $K \geq 4N$, and N is the number of antennas. The objective of the data matrix methods is to find the weight vector $\vec{w}$ that minimizes the norm of the error vector, $\vec{e} = \vec{y} - \vec{d} = \vec{w}^H X - \vec{d}$ [15] usually in the least squares sense. Thus the weight vector $\vec{w}$ is the solution to the problem $\min_{\vec{w}} \sum_{k=0}^{K-1} |\vec{w}^H \vec{x}_k - d_k|^2$.

Data matrix based algorithms which have been used include the singular value decomposition (SVD) of X, QR factorization of X by Givens rotations [6], Householder transformation [12], Gram-Schmidt [2] and modified Gram-Schmidt (MGS) orthogonalization and divisionless MGS [17].

1.4 Gradient based algorithms

Gradient-based minimization methods have also been used to solve the beamforming problem. In equation (1.2.1), we see that the MSE is given by a positive definite quadratic form. Its N-dimensional surface is therefore described by a paraboloid and the desired solution is at the minimum of this function. The simplest gradient technique for minimization is the steepest descent method which for this problem can be described, following [5, p.198] as follows:

The iteration proceeds as

$$\vec{w}_{k+1} = \vec{w}_k - \frac{\mu}{2}\nabla\overline{e_k^2}$$

where $\vec{w}_k$ is the estimated weight vector at time k and $\vec{w}_{k+1}$ is the new estimate at time $k+1$, and μ is the constant step size. The gradient vector is given by equation (1.2.2) and thus we have the iteration

$$\vec{w}_{k+1} = \vec{w}_k + \mu(\vec{r}_{xd} - R_{xx}\vec{w}_k)$$

In this particular context, an even simpler iteration for the weight vectors is achieved in the Least Mean Squares (LMS) method [5, p.217] by dropping the expectation operator in the MSE equations. The gradient vector is then approximated by $2e_k\vec{x}_k$ so that the iteration reduces to just

$$\vec{w}_{k+1} = \vec{w}_k + \mu\vec{x}_k e_k$$

where the step size satisfies $0 < \mu < 2/\lambda_{max}$ and λ_{max} is the largest eigenvalue of R_{xx}.

2. Residue Number Systems

2.1 Introduction

Residue Number System (RNS) arithmetic has been considered as a number representation for digital computers since the early days of developing computers. A residue number system, is an exact arithmetic using the integers, **Z**. An RNS is a parallel carry-free arithmetic. Due to this parallelism, the residue number system can perform addition and multiplication very fast compared to conventional integer processors without sacrificing dynamic range. The parallel channels provide inherent fault tolerance, by using redundancy. A Texas Instrument study showed that a FIR filter designed using RNS arithmetic is expected to have a high speed-to-area ratio and a high speed-to-power ratio compared to a binary implementation of the same filter. These results are expected for other multiply-accumulate intensive problems.

Because RNS is restricted to the integers, it cannot be used as the sole arithmetic in general purpose computers. Implied by this constraint is the requirement to convert between RNS arithmetic and standard binary arithmetic. This conversion is required to perform operations that can not be performed efficiently in RNS such as division, square roots and magnitude comparison. The integers are not closed under division, for example, so RNS can not readily be used for division. Whenever it is necessary to perform a division (or even when the algorithm has been cleaned of all overt divides, the scaling of data requires division) conversion to a weighted code such as binary is necessary.

Signal processing tasks such as FIR filtering and DFTs are multiply-accumulate (MAC) intensive operations, hence RNS is an ideal arithmetic for these types of operations. More complicated algorithms such as adaptive processing, present the more difficult operations for RNS. The conversion of RNS numbers to their weighted system equivalent is very expensive and if a sufficient number of conversions must be made, the advantages of RNS are outweighed. Unfortunately, because of the need for scaling and magnitude information in many signal processing algorithms, these conversion costs hindered the development of RNS signal processors. Much of the rest of this paper is concerned with how we overcome the need for some of these operations within the adaptive beamforming algorithm. We also discuss some implementation details.

For an extensive introduction to RNS see [13].

2.2 Theory and Examples

Integers are mapped to an L-tuple of residues by reducing the integer mod p_i ($1 \le i \le L$) where the *moduli* p_i are relatively prime integers. The dynamic range, M, of the system is the product of the moduli:

$$M = \prod_{i=1}^{L} p_i$$

Arithmetic operations are performed on the respective elements in the L-tuples. For example, integers X and Y are mapped to $(x_1, x_2, \ldots x_L)$ and $(y_1, y_2, \ldots y_L)$ where $x_i = X \bmod p_i$, $y_i = Y \bmod p_i$. Addition and multiplication are then performed by componentwise modular arithmetic:

$$X + Y = \left(\langle x_1 \oplus y_1\rangle_{p_1}, \langle x_2 \oplus y_2\rangle_{p_2}, \cdots, x_L \oplus y_L\rangle_{p_L}\right)$$

$$X \times Y = \left(\langle x_1 \otimes y_1\rangle_{p_1}, \langle x_2 \otimes y_2\rangle_{p_2}, \cdots, x_L \otimes y_L\rangle_{p_L}\right)$$

where the notation $\langle a \oplus b\rangle_p$, $\langle a \otimes b\rangle_p$ denote the arithmetic operations mod p.

A simple example follows.

Example 1

The summation of 34 and 54 using the moduli 3, 5, 7 is summarized by the diagram:

			Moduli			
			3	5	7	
Operand 1	34	→	1	4	6	Residues
Operand 2	54	→	0	4	5	Residues
Sum	88	←	1	3	4	Residues

The two operands are converted to their RNS representations by storing their remainders (or *residues*) after division by the respective moduli. In a practical system this conversion would be done by a multi-stage look-up table. The input value is separated into a set of *subwords* each representing a partial sum of the digits of that number. The residue, mod p, say, of each subword can be computed independently and the results then added, alos mod p, to obtain the desired residue.

In this example, the residues are then added componentwise relative to the appropriate moudulus: $1+0 \equiv 1 \bmod 3$, $4+4 \equiv 3 \bmod 5$, and $6+5 \equiv 4 \bmod 7$.

The resulting L-tuple which is the RNS representation of the sum can, if desired, be converted back to standard representation using the Chinese Remainder Theorem. Remainder Theorem. For this conversion, we need the quantities $m_i = M/p_i$, and their mod p_i reciprocals, $\langle m_i^{-1}\rangle_{p_i}$ for $i=1,2,3,...,L$. The inverse mapping $X = \phi^{-1}(x_1, x_2, ..., x_L)$ is then given by

$$X = \left\langle \sum_{i=1}^{L} m_i \langle m_i^{-1} x_i \rangle_{p_i} \right\rangle_M$$

Example 2

Let $p_1=3, p_2=5, p_3=7$. Then $M = p_1p_2p_3 = 105$. Let $a = 7$, and $b = 9$ where $a, b \in Z_M$.

$$\phi(a) = (\langle 7\rangle_3, \langle 7\rangle_5, \langle 7\rangle_7) = (1,2,0), \quad \phi(b) = (\langle 9\rangle_3, \langle 9\rangle_5, \langle 9\rangle_7) = (0,4,2)$$

Then, for example,

$$(1,2,0) + (0,4,2) = (\langle 1+0\rangle_3, \langle 2+4\rangle_5, \langle 0+2\rangle_7) = (1,1,2)$$

$$(1,2,0) \times (0,4,2) = (\langle 1\times 0\rangle_3, \langle 2\times 4\rangle_5, \langle 0\times 2\rangle_7) = (0,3,0)$$

The inverse mapping requires the following values, $m_1=35$, $m_2=21$, $m_3=15$, $\langle m_1^{-1}\rangle_3 = 2$, $\langle m_2^{-1}\rangle_5 = 1$, $\langle m_3^{-1}\rangle_7 = 1$ then, using the CRT we get

$$\phi^{-1}(1,1,2) = \left\langle \sum_{i=1}^{3} m_i \langle m_i^{-1} x_i \rangle_{p_i} \right\rangle_{105} = \langle 35\langle 2\times 1\rangle_3 + 21\langle 1\times 1\rangle_5 + 15\langle 1\times 2\rangle_7\rangle_{105} = 16$$

and

$$\phi^{-1}(0,3,0) = \left\langle \sum_{i=1}^{3} m_i \langle m_i^{-1} x_i \rangle_{p_i} \right\rangle_{105} = \langle 35\langle 2\times 0\rangle_3 + 21\langle 1\times 3\rangle_5 + 15\langle 1\times 0\rangle_7\rangle_{105} = 63$$

An extension to RNS is the Quadratic RNS (QRNS) [1] which allows complex integer arithmetic using pairs of real integers. For example, a complex integer $(a+jb)$ is mapped to a pair of real integers (z,z^*). Given a prime p of the form $p = 4k+1$ where $k \in \mathbf{Z}$, *Gaussian primes*, then the congruence $x^2 = -1 \text{ mod } p$ has two solutions in the field $\mathbf{Z}_p$ that are multiplicative and additive inverses of one another. Let $\hat{j}$, $\hat{j}^{-1}$ denote these two solutions. Define a mapping from the complex integers mod p, $\mathbf{Z}_p[j]$ into $\mathbf{Z}_p \times \mathbf{Z}_p$ by

$$\theta(a+jb) = (z, z^*)$$

where

$$z = (a+jb) \text{ mod } p, \qquad z^* = (a-jb) \text{ mod } p$$

The inverse mapping is given by

$$\theta^{-1}(z, z^*) = \langle 2^{-1}(z+z^*)\rangle_p + j\langle 2^{-1}\hat{j}^{-1}(z-z^*)\rangle_p$$

Addition is preserved by this map. That is, if $\theta(a+jb) = (z, z^*)$, $\theta(c+jd) = (w, w^*)$ then $\theta((a+c)+j(b+d)) = (z+w, z^*+w^*)$ where the additions are performed with respect to a given modulus.

The following are suitable Gaussian primes of varying binary lengths:

up to 7-bits

{5, 13, 17, 29, 37, 41, 53, 61, 73, 89, 97, 101, 109, 113}

8-bits

{137, 149, 157, 173, 181, 193, 197, 229, 233, 241}

9-bits

{257, 269, 277, 281, 293, 313, 317, 337, 349, 353, 373, 389, 397, 401, 409, 421, 433, 449, 457, 461, 509}

10-bits

{521, 541, 557, 569, 577, 593, 601, 613, 617, 641, 653, 661, 673, 677, 701, 709, 733, 757, 761, 769, 773, 797, 809, 821, 829, 853, 857, 877, 881, 929, 937, 941, 953, 977, 997, 1009, 1013, 1021}

Example 3

Let $p = 101, x = (34 + 63j)$. We want to find a j such that $\langle j^2\rangle_{101} = \langle -1\rangle_{101} = \langle 100\rangle_{101}$. Clearly, $j = 10$ is one solution. Also, the other solution is $j^{-1} = 91$ which is both an additive and multiplicative inverse of j:

$$\langle j + j^{-1}\rangle_{101} = \langle 10 + 91\rangle_{101} = \langle 101\rangle_{101} = 0$$

$$\langle j \times j^{-1}\rangle_{101} = \langle 10 \times 91\rangle_{101} = \langle 910\rangle_{101} = 1$$

The QRNS pairs are then computed as:

$$\theta(34+63j) = \{\langle 34+10(63)\rangle_{101}, \langle 34-10(63)\rangle_{101}\} = \{58, 10\}$$

To invert this QRNS process we use the fact that

$$Re(z) = (z+\bar{z})/2, \quad Im(z) = (z-\bar{z})/2j$$

and so we need $\langle 2^{-1}\rangle_{101} = 51$ and then $\langle (2j)^{-1}\rangle_{101} = \langle (51)(91)\rangle_{101} = 96$. Then

$$\theta^{-1}(58,10) = \langle 51(58+10)\rangle_{101} + j\langle 96(58-10)\rangle_{101} = 34 + 63j$$

as expected.

In a conventional arithmetic unit, a complex multiplication requires four real multiplies and two real adds. A further extension to QRNS is the Galois Enhanced QRNS (GEQRNS) [1] which allows this operation count to be reduced to just two real RNS-adds. To achieve this, we map the pair (z, z^*) to their logarithms with respect to a generator of $\mathbf{Z}_p$, (e_z, e_{z^*}). For any prime modulus p there exists some $\alpha \in \mathbf{Z}_p$ that generates all non-zero elements of the field $GF(p)=\mathbf{Z}_p$. That is to say $\{\alpha^i \mid i=0,1,2,\ldots,p-2\} = GF(p)\backslash 0$. The integer z is therefore equivalent to $\langle \alpha^{e_z}\rangle_p$ and can be uniquely represented by this exponent. These number theoretic logarithms may be added modulo $p-1$ to produce multiplications:

$\langle \alpha^{\langle i+j\rangle_{p-1}}\rangle_p = \langle \alpha^i \alpha^j\rangle_p$. Hence, a complex multiply requires just 2 real adds. Since zero cannot be generated, zero must be detected and handled as a special case.

Example 4

As an example of GEQRNS arithmetic, let $p = 7$, then a generator is $\alpha=3$ so that $GF(7)\backslash 0 = \{\langle 3^i\rangle_7 \mid i = 0,1,2,3,4,5\} = \{1,3,2,6,4,5\}$. To multiply 3 and 5, we simply add (modulo 6) their mod 7 logarithms (to base $\alpha=3$):

$$3^5 = 5 \bmod 7 \rightarrow \log_3(5) = 5$$

$$3^1 = 3 \bmod 7 \rightarrow \log_3(3) = 1$$

so that

$$\langle 5\times 3\rangle_7 = \langle 3^5\times 3^1\rangle_7 = \langle 3^{\langle 5+1\rangle_6}\rangle_7 = \langle 3^0\rangle_7 = 1$$

2.3 Scaling in RNS

In a practical system, RNS does not have infinite precision. There is a dynamic range limitation; just as there is a limitation in a conventional integer processor. When designing algorithms for RNS implementation, think of the RNS processor as an integer processor with constraints, such as division and square root operations are not close in the integers, therefore there is no simple way that these operations can be done in RNS. An integer processor on the other hand can approximate these operations since rounding may occur, but rounding can't happen in RNS.

Algorithms must be designed to keep the growth of intermediate results under control, as in any algorithm design on a conventional integer processor. Periodic scaling is necessary to keep the growth under control. Unfortunately, the scaling can not be done directly in the RNS. Scaling can be accomplished by converting back to the integers and then dividing. The conversion, through the Chinese Remainder Theorem (CRT) or the Mixed Radix Conversion (MRC) algorithms, or through the core function [13], requires extra overhead that may negate the advantages of the RNS. Therefore, the object is to stay in the RNS as long as possible, being careful not to overflow the dynamic range of the system.

How do you check if you are about to overflow? In conventional processors, the intermediate results can be compared to the maximum number that can be represented on the processor. If that result is approaching the upper limit, then it is time to scale, keeping track of the scale factor to reconstruct the proper result (if necessary) later. Unfortunately, the comparison operation can not easily be done in RNS either. Because of this problem, the term growth must be analyzed, or it must be determined if the problem can tolerate this fatal overflow error occasionally. It may turn out that the fatal errors may occur infrequently and can be tolerated.

An analysis of the element growth is needed in order to determine how many operations can be performed before scaling is required in order to avoid overflowing the available dynamic range. Of course, more frequent scaling results in greater loss of precision and, as we see shortly, the later we scale the better in terms of final accuracy. It is therefore better to scale occasionally during the computation rather than to prescale the input data.

Another possibility is continuous scaling which can be accomplished (at the cost of some loss of precision) via the *L*-CRT. The advantage is that this lends itslef to a data-flow architecture. The *L*-CRT operation can be partitioned into 4 stages [4], [7]. When one stage is complete, it passes the result to another stage. The output is delayed until the data passes through all of the stages in the pipe. This delay is called pipeline latency. If a continuous stream of data needs to be converted from RNS to binary, the effective conversion rate is one conversion every clock cycle, but it still takes 4 clock cycles to perform a single conversion. With this in mind, it may be possible to take advantage of this pipelining in the algorithm. The algorithm may scale continuously, using the L-CRT, instead of occasional scaling.

The L-CRT is computed by factoring M into a real scale factor V and an integer $M' = 2^k$, where $k \epsilon \mathbf{Z}^+$, such that $M = V M'$, and $0 < M' < M$. The L-CRT is given by

$$X_S = \left\langle \sum_{i=1}^{L} \left\lfloor m_i \langle m_i^{-1} x_i \rangle_{p_i} / V \right\rfloor \right\rangle_{M'}$$

where $\lfloor \bullet \rfloor$ denotes the integer-part or floor function and $m_i = \prod_{\substack{j=1 \\ j \neq i}}^{L} p_j$. The L-CRT is a residue-to-binary conversion that automatically scales by V. The disadvantage of the L-CRT is that it may introduce an error into the computed X_S. The error in the L-CRT is given by $0 \leq | X/V - X_S | < L$ which is usually small since $L \ll M$.

To halve the word length, the scale factor is on the order of $\sqrt{M}$. For the system using $\{p_1, p_2, p_3\} = \{101,109,113\}$, $M = 1244017$, $\sqrt{M} = 1115.3551$ which corresponds to about $k=10$ or about half of the 20.2 bits of M.

The scale factors V for various values of k for this system are:

k	$M' = 2^k$	Scale Factor $V=M/M'$	k	$M' = 2^k$	Scale Factor $V=M/M'$
0	1	1244017	10	1024	1214.860352
1	2	622008.5	11	2048	607.4301758
2	4	311004.25	12	4096	303.7150879
3	8	155502.125	13	8192	151.8575439
4	16	77751.0625	14	16384	75.92877197
5	32	38875.53125	15	32768	37.96438599
6	64	19437.76563	16	65536	18.98219299
7	128	9718.882813	17	131072	9.491096497
8	256	4859.441406	18	262144	4.745548248
9	512	2429.720703	19	524288	2.372774124
			20	1048576	1.186387062

3. Gaussian elimination

3.1 Why Gauss elimination?

The primary reason for selecting Gauss elimination as our base algorithm for the solution of the linear system is that it provides a simple algorithm using fewer arithmetic operations than most other options. The drawback of Gauss elimination is that the factors of the matrix are not retained and so repeated solution with different right hand sides is not facilitated.

In our beamforming application however, we face "one-off" problems so that retention of the matrix factors is not so important. Any change in the relative directions or relative strengths of the required signal and jammer results in a new system.

Gauss elimination demands relatively few non-RNS operations (such as divisions and square roots) and the divisions that are needed can be delayed or even eliminated by modification of the algorithm. Unlike symmetric factorizations such as Cholesky or direct solution of the least squares problem using QR factorization, *no* square-root operations are necessary.

The modified algorithm discussed here uses integer arithmetic performed in RNS with some scaling and range extension to achieve a solution which is a scalar multiple of the desired solution. Since only the relative magnitudes of the components of the solution are significant this is sufficient.

In its most basic form Gauss elimination for the solution of an $n \times n$ system

$$A\vec{x} = \vec{b}$$

can be described as an $n-1$ step process in which, at stage i we eliminate all entries in column i below the main diagonal of the matrix by performing elementary row operations on the matrix. Of course, whatever row operations are performed on the matrix must also be applied to the right hand side.

We denote the matrix at the i^{th} stage by $A^{(i)}$ (so that $A = A^{(1)}$) and its elements by $a_{jk}^{(i)}$. Similarly the components of the right-hand side at stage i will be denoted by $b_j^{(i)}$. In its simplest form (often termed the *ijk*-form) the Gauss elimination algorithm is then

```
for i=1 to n-1
    for j=i+1 to n
        $m = a_{ji}^{(i)} / a_{ii}^{(i)}$,    $a_{ji}^{(i+1)} = 0$
        $b_j^{(i+1)} = b_j^{(i)} - m b_i^{(i)}$
        for k=i+1 to n
            $a_{jk}^{(i+1)} = a_{jk}^{(i)} - m a_{ik}^{(i)}$
```

To complete the solution, this elimination phase is followed by the back substitution:

```
$x_n = b_n^{(n)} / a_{nn}^{(n)}$
for i=n-1 down to 1
    for j=i+1 to n
        $b_i^{(i)} = b_i^{(i)} - a_{ij}^{(i)} x_j$
    $x_i = b_i^{(i)} / a_{ii}^{(i)}$
```

For general linear systems, pivoting is necessary in Gauss elimination in order to reduce the effect of roundoff error and preserve maximal accuracy in the computed result. Why are we not proposing to use pivoting here? The first reason is that integer arithmetic is being used and so roundoff error is not an issue. (Actually, there is some loss of precision involved in our solution by virtue of the scaling which is used to a limited extent in the modified algorithm presented. This will be discussed in detail later.) The matrix here is hermitian positive definite. The use of partial pivoting destroys the symmetry and therefore the ability to economize on storage if that becomes an issue. Also, and more importantly, for a positive definite system using partial pivoting does not improve the numerical stability of the algorithm. [16]

3.2 Avoidance/ elimination of the divisions

In order to maximize the efficiency of our RNS processors for the solution of the linear system, we want to delay or even eliminate the non-RNS operations (divisions in the case of Gauss elimination). At this stage we can simply regard this as being a requirement to achieve the solution on an integer processor.

Consider therefore one step of the elimination process. Suppose then that we are eliminating in column i and consider the effect of this elimination on row $j>i$. In the conventional application of Gauss elimination, we use the multiplier $a_{ji}^{(i)}/a_{ii}^{(i)}$ so that each subsequent member of row j is replaced by $a_{jk}^{(i+1)} = a_{jk}^{(i)} - a_{ik}^{(i)}\left(a_{ji}^{(i)}/a_{ii}^{(i)}\right)$. Clearly this requires a noninteger operation.

This division can be eliminated by simply "cross-multiplying" between the two rows so that for each $k>i$

$$a_{jk}^{(i+1)} = a_{ii}^{(i)}a_{jk}^{(i)} - a_{ji}^{(i)}a_{ik}^{(i)}$$

This evidently preserves the integer nature of the matrix elements but has associated computational costs. The most important difficulty introduced by this integer arithmetic requirement is that the size of matrix elements can grow rapidly as the elimination phase progresses.

3.3 Growth of the matrix elements

To get an idea of the rate of growth which may be encountered, consider just one step of the elimination in which we are effectively dealing with a 2×2 matrix. The standard Gauss elimination results in the modification:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow \begin{bmatrix} a & b \\ 0 & d-b(c/a) \end{bmatrix}$$

whereas the integer preserving form yields:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow \begin{bmatrix} a & b \\ 0 & ad-bc \end{bmatrix}$$

in which the bottom-right element is a times that for the standard algorithm. For the full elimination this results in the final element $a_{nn}^{(n-1)}$ becoming the full determinant of the original matrix - potentially a large number.

At each stage of the elimination, there is the possibility that the largest element in the matrix could approach twice the square of the largest element at the previous stage.

Example 5

By way of illustration consider just a simple 3×3 example. The elimination procedure for the following matrix yields

$$\begin{bmatrix}7 & 8 & 9\\ 8 & 3 & 2\\ 9 & 2 & 7\end{bmatrix} \rightarrow \begin{bmatrix}7 & 8 & 9\\ 0 & -43 & -58\\ 0 & -58 & -32\end{bmatrix} \rightarrow \begin{bmatrix}7 & 8 & 9\\ 0 & -43 & -58\\ 0 & 0 & -1988\end{bmatrix}$$

We see that although the growth here is not as severe as the worst case described above, there is still very rapid growth giving a greatest element close to $(1/3)9^4$.

In our beamforming problem, we would be performing complex integer arithmetic which allows the possibility of even (slightly) faster growth in the matrix elements. For this approach to be viable we must clearly be able to handle a very large dynamic range in the later stages of the elimination. This can be achieved in principle by the "column-parallel, parallel-channel" approach described below.

3.4 Column-parallel, parallel-channel implementation

The basic idea is to use a parallel array of RNS processors which are allocated to the various columns of the matrix. As the elimination proceeds, fewer columns are still "active". The processors used for the inactive columns can be reallocated to extend the dynamic range available for the remaining columns.

We begin by recalling briefly the column parallel version of the Gauss elimination algorithm. Denote by $\vec{a}_j^{(i)}$ that part of the jth column of $A^{(i)}$ below row i. The basic Gauss elimination algorithm entails the formation of the vector $\vec{m}^{(i)} = \vec{a}_i^{(i)}/a_{ii}^{(i)}$ and then each subsequent column is modified using vector the operation

$$\vec{a}_j^{(i)} = \vec{a}_j^{(i)} - a_{ij}^{(i)}\vec{m}^{(i)}$$

This is just the standard vector processor operation "vector + scalar × vector". The obvious modification for our integer algorithm is

$$\vec{a}_j^{(i)} = a_{ii}^{(i)}\vec{a}_j^{(i)} - a_{ij}^{(i)}\vec{a}_i^{(i)}$$

which is a similar though slightly extended "scalar × vector + scalar × vector" operation for which our processor can be suitably designed.

The idea behind the adaptive parallel-channel approach to implementing this integer algorithm in RNS arithmetic is that a number of parallel RNS processor channels would be used, each operating relative to a specific modulus. The number of such channels allocated to a particular data item essentially determines the dynamic range available for that data. Initially the processors would be divided evenly among the columns of the matrix.

After the first stage of the elimination the number of active columns is reduced by

1 but the necessary dynamic range is increased. The idea is to adaptively allocate processors to columns so that as the required dynamic range grows, the number of processors grows too. The basic idea is easily illustrated for the case of four columns using a total of 12 RNS processor channels.

The adaptive nature of the algorithm is illustrated in Figure 3 below.

FIGURE 3

Schematic diagram of the adaptive dynamic range allocation of RNS processors.

Active matrix	Processor											
	1	2	3	4	5	6	7	8	9	10	11	12
	RNS basis vector and residues of matrix elements											
$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} \\ a_{41}^{(1)} & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} \end{bmatrix}$	p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3
	$a_{11}^{(1)}$	$a_{11}^{(1)}$	$a_{11}^{(1)}$	$a_{12}^{(1)}$	$a_{12}^{(1)}$	$a_{12}^{(1)}$	$a_{13}^{(1)}$	$a_{13}^{(1)}$	$a_{13}^{(1)}$	$a_{14}^{(1)}$	$a_{14}^{(1)}$	$a_{14}^{(1)}$
	$a_{21}^{(1)}$	$a_{21}^{(1)}$	$a_{21}^{(1)}$	$a_{22}^{(1)}$	$a_{22}^{(1)}$	$a_{22}^{(1)}$	$a_{23}^{(1)}$	$a_{23}^{(1)}$	$a_{23}^{(1)}$	$a_{24}^{(1)}$	$a_{24}^{(1)}$	$a_{24}^{(1)}$
	$a_{31}^{(1)}$	$a_{31}^{(1)}$	$a_{31}^{(1)}$	$a_{32}^{(1)}$	$a_{32}^{(1)}$	$a_{32}^{(1)}$	$a_{33}^{(1)}$	$a_{33}^{(1)}$	$a_{33}^{(1)}$	$a_{34}^{(1)}$	$a_{34}^{(1)}$	$a_{34}^{(1)}$
	$a_{41}^{(1)}$	$a_{41}^{(1)}$	$a_{41}^{(1)}$	$a_{42}^{(1)}$	$a_{42}^{(1)}$	$a_{42}^{(1)}$	$a_{43}^{(1)}$	$a_{43}^{(1)}$	$a_{43}^{(1)}$	$a_{44}^{(1)}$	$a_{44}^{(1)}$	$a_{44}^{(1)}$
$\begin{bmatrix} a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{bmatrix}$	p_1	p_2	p_3	p_4	p_1	p_2	p_3	p_4	p_1	p_2	p_3	p_4
	$a_{22}^{(2)}$	$a_{22}^{(2)}$	$a_{22}^{(2)}$	$a_{22}^{(2)}$	$a_{23}^{(2)}$	$a_{23}^{(2)}$	$a_{23}^{(2)}$	$a_{23}^{(2)}$	$a_{24}^{(2)}$	$a_{24}^{(2)}$	$a_{24}^{(2)}$	$a_{24}^{(2)}$
	$a_{32}^{(2)}$	$a_{32}^{(2)}$	$a_{32}^{(2)}$	$a_{32}^{(2)}$	$a_{33}^{(2)}$	$a_{33}^{(2)}$	$a_{33}^{(2)}$	$a_{33}^{(2)}$	$a_{34}^{(2)}$	$a_{34}^{(2)}$	$a_{34}^{(2)}$	$a_{34}^{(2)}$
	$a_{42}^{(2)}$	$a_{42}^{(2)}$	$a_{42}^{(2)}$	$a_{42}^{(2)}$	$a_{43}^{(2)}$	$a_{43}^{(2)}$	$a_{43}^{(2)}$	$a_{43}^{(2)}$	$a_{44}^{(2)}$	$a_{44}^{(2)}$	$a_{44}^{(2)}$	$a_{44}^{(2)}$
$\begin{bmatrix} a_{33}^{(3)} & a_{34}^{(3)} \\ a_{43}^{(3)} & a_{44}^{(3)} \end{bmatrix}$	p_1	p_2	p_3	p_4	p_5	p_6	p_1	p_2	p_3	p_4	p_5	p_6
	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$
	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{42}^{(3)}$	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$

In Figure 3, for a hypothetical situation of a 4×4 system, the first stage uses a dynamic range covered by a three-dimensional RNS representation. At the next stage, only three columns are active and so a fourth basis element can be introduced extending the dynamic range. Similarly for the third (and final) stage of the forward elimination only two columns remain active and so a six-dimensional representation can be used. The extent of the range extension is obviously dependent on the relative sizes of the basis elements.

For the first stage of the back substitution, all 12 channels could be used to accommodate a greatly increased dynamic range which could then be reduced as the solution process proceeds. However, if the divisions required in the back substitution are also to be postponed or eliminated, then the dynamic range will grow further during that phase of the solution process. This is discussed later.

If the basis elements are all of similar magnitudes and the initial range is appropriate to the covariance matrix then the growth allowed for here is clearly insufficient for the worst case growth discussed above. The description and figure are intended to convey the broad philosophy of the solution process not the practical detail. The number of RNS channels, the choice of basis elements and control of the growth of matrix elements are all important factors to be discussed shortly.

One important consideration which is suggested by the schematic diagram concerns the possibility of changing the prime modulus used by a specific channel during the computation. An RNS processor could be designed either to have a fixed modulus or to be programmable in the sense that the modulus can be changed during the computation.

If the base modulus of a processor is fixed throughout and all of them are required for the final stage then, for conventional RNS arithmetic, all the moduli must be distinct. This presents potential problems since there may be insufficient suitable moduli of sufficient magnitude (but still representable in only, say, 8 bits) available for use. The solution process would also require frequent conversions of RNS representations between different basis vectors. Although this is an achievable task it necessarily costs time.

An alternative within the fixed modulus framework might be to use repeated moduli and a mixed radix arithmetic system.

With programmable moduli, most of the above difficulties would be alleviated although there would remain problems of base-extension as the dynamic range and therefore the dimension of the basis increases. However the same basis vector can then be used throughout any stage of the elimination and this can always include the previous basis. Of course there are costs associated with changing the base modulus and these must be weighed against other costs of the overall solution process.

It is important for understanding the overall algorithm to appreciate that Figure 3 should not be interpreted too literally. The apparent reprogramming of processors 4 through 12 for the second stage is wasteful. The process is pictured that way for simplicity. In practice we should anticipate processors 1 through 9 being unchanged - and used for the base extension to be discussed shortly - while processors 10 through 12 are all modified for modulus p_4. Column 2 would then be processed using channels 1, 2, 3 and 10, while column 3 would use 4, 5, 6 and 11, and column 4 uses 7, 8, 9 and 12. Similar modifications would take place at subsequent stages.

3.5 Back substitution

The back substitution phase of the solution begins with the single equation

$$a_{nn}^{(n-1)} w_n = s_n^{(n-1)}$$

which appears to require division immediately. However, we are only interested in the *relative* sizes of the weights and so this division need not be performed at this stage. However this implies a need for a further "cross-multiplication" at the next and subsequent stages of the solution.

This in turn suggests further growth in the dynamic range required to accommodate the computation. If no scaling has been done in the forward elimination phase, then the dynamic effective wordlength doubled at each step. It follows that in the back substitution the effective wordlength required would increase more slowly since the ranges of the multipliers are reducing. The overall effect would be that one further doubling of the wordlength would suffice for the whole back substitution phase.

4. Difficulties with the proposed algorithm

There are several difficulties already apparent in the outlined solution process. In this section we discuss some of these in detail and present refinements of the algorithm to alleviate these problems. We discuss these within the specific framework of a four antenna array, so that the covariance matrix will be 4×4. It will also be assumed that the elements of the initial covariance matrix can be uniquely represented using three 7-bit moduli. The overall array of RNS processors will be assumed to have 16 such processors. Of course whatever operations are performed on the matrix must also be performed with the right hand side vector. We shall assume there is a similar array of 16 processors performing those operations in parallel with the matrix operations.

4.1 Obtaining the covariance matrix

The elements of the covariance matrix are formed as scalar products of the data vectors. These, for a model 4-antenna problem, would typically be vectors of length around 16.

Scalar products are readily performed in the proposed RNS architecture and so obtaining the covariance matrix relative to a fixed RNS basis is straightforward provided that the available dynamic range is sufficient to accommodate the intermediate results.

Using the 16 processors subdivided into 4 groups of four each representing a column of the covariance matrix, we can compute a complete row of this matrix in parallel in a single RNS-channel scalar product time. Four such operations are therefore sufficient to generate the complete array. Note that at this stage we have a 4-dimensional basis vector even though the matrix elements could be stored using just three base moduli. The additional channel per column is essentially free at this stage but accommodates some of the dynamic range growth that will be needed for subsequent stages.

Implicitly, we have assumed here that the individual RNS processors are programmable for different base moduli so that the same basis vector can be used for representation of all columns. Of course any changes of base modulus carry a penalty in the need to load new arithmetic tables. This time penalty must be balanced against any loss of precision which may be entailed in controlling the dynamic range.

Scaling of the initial data or of the covariance matrix could be used to reduce the dynamic range requirement at the outset. This is equivalent to coarsening the resolution of the input data. We consider the effect of such a reduction in resolution shortly in connection with the question of scaling during the forward elimination. If any such scaling is to be used then clearly the choice of scale factors will be an important consideration.

4.2 Element growth

Almost obviously, the biggest problem with the proposed solution technique is the rate of growth of the matrix elements and the consequent range extension requirement. In the case of a positive definite matrix, Wilkinson [16] establishes that the growth in Gauss elimination using divisions is bounded by 1. That is there is no growth. However if integer arithmetic without divisions is used, it is clear that the elements can grow rapidly. As we showed in the previous section the rate of growth can be of the order of squaring the largest element at each stage of the elimination.

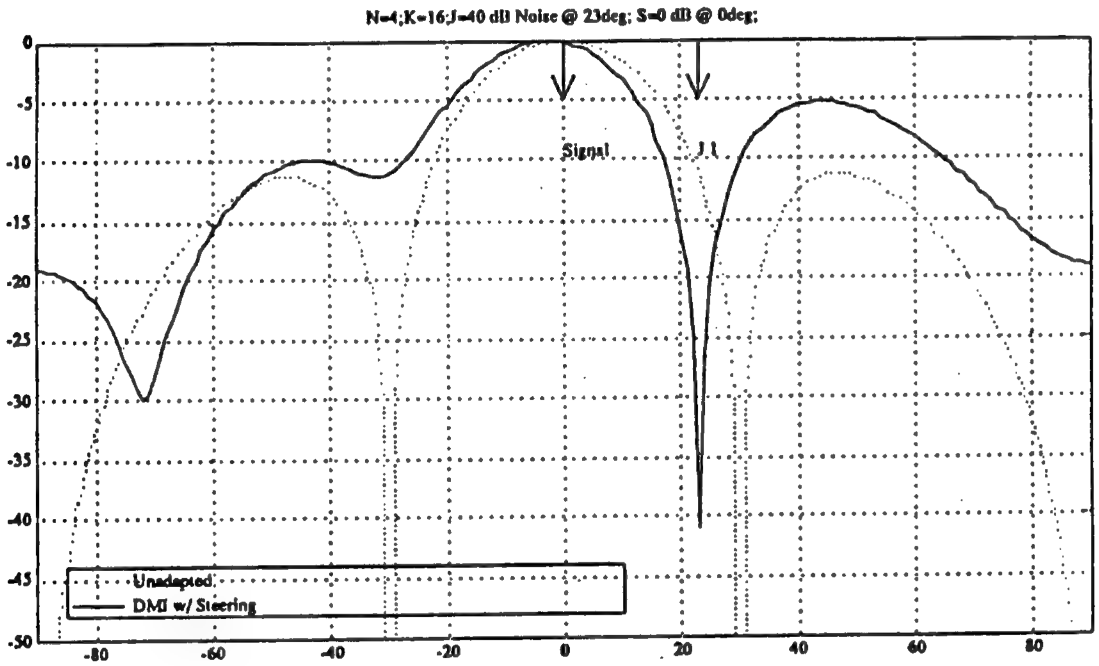
By way of illustration, we consider one example.

Example 6

Consider the situation of N=4 antenna, K=16 data vectors for a signal S=0 dB at 0° and a jammer J=40 dB noise at 23°. The unadapted and adapted beam patterns for this problem are shown in Figure 4.

FIGURE 4

Unadapted and adapted beam patterns for Example 6



The averaging in the covariance matrix entails a division of the scalar products by K but since the relative weights are the required quantities this division can be ignored in the solution process. For this case the resulting matrix is

$$\begin{bmatrix} 254879 & 84113-240210j & -200191-159002j & -220468+135323j \\ 84113+240210j & 254223 & 83794-241184j & -200316-163148j \\ -200191+159002j & 83794+241184j & 256489 & 88751-243850j \\ -220468-135323j & -200316+163148j & 88751+243850j & 262597 \end{bmatrix}$$

The largest element in the initial matrix is around 2.6×10^5 which is close to 2^{18}. It is therefore uniquely representable in the proposed 3-dimensional RNS form. After one stage of the elimination the active matrix is

$$\begin{bmatrix} 20263148 & 2126089-10721400j & -6179050-8804311j \\ 2126089+10721400j & 15587346 & 1684387-6944521j \\ -6179050+8804311j & 1684387+6944521j & 12007410 \end{bmatrix}$$

which now has a largest element of around 2×10^7 or 2^{24}. The growth at this first stage has been substantially less than the worst case mentioned above. A 4-dimensional RNS representation would be sufficient in this case. Of course, we have availability for a 5-dimensional representation at this stage using our 16 RNS channels.

The next stage of the elimination reduces the active matrix to the 2×2 system

$$\begin{bmatrix} 1.9638\times10^{14} & -2.568\times10^{13}-6.000\times10^{13}j \\ -2.568\times10^{13}+6.000\times10^{13}j & 1.5883\times10^{14} \end{bmatrix}$$

in which the largest element is now around 2×10^{14} so that almost worst case growth has indeed occurred here. An 7-dimensional representation would now be necessary. (Eight channels per column are now available.) The final elimination yields the final element 2.693×10^{28} so that near worst case growth has again occurred but the 16 available RNS channels can easily accommodate this range.

Of course the growth of the elements in this example cannot be assumed to represent the general case. In the absence of any special knowledge we must allow for worst case growth. Such growth could not be accommodated in the same array as described above without some other means of controlling that growth. One possibility would be to scale the active matrix periodically. Before discussing the use of scaling, we must consider the questions arising out of the base-extensions.

4.3 Base extension

There are two major aspects to the problems posed by the base extensions required by the algorithm under consideration. The first is the mathematical problem of finding the residues relative to new basis elements of an integer given only by its residues relative to the existing basis. This question and variations of it have been discussed extensively for various special cases [3], [10], [11], [14], for example.

The case of interest here is almost the simplest in that all we wish to accomplish is the addition of one or more new moduli to the basis. Within our parallel architecture, any one processor would be concerned solely with the addition of a single basis element. The process is simply described in [3] by conversion from the existing standard RNS representation to the associated mixed radix system, MRS - an operation which is easily achieved using residue arithmetic throughout - and then computing the residue of the resulting mixed radix representation relative to the new basis element.

A simple example is included for completeness.

Example 7

Suppose we have an integer $n \in [0, 105]$ represented in the RNS system with basis $\{3,5,7\}$ by the vector $(1,3,3)$.

We must first convert this into mixed radix form relative to this same basis. That is we must find $a_0 \in \{0,1,2\}, a_1 \in \{0,...,4\}, a_2 \in \{0,...,6\}$ satisfying $n = a_0 + a_1(3) + a_2(3)(5)$ It is immediate that $a_0 = |n|_3 = 1$ then $a_1 = |3^{-1}(n-a_0)|_5 = |2(3-1)|_5 = 4$ and finally
$$a_2 = |3^{-1}5^{-1}(n-a_0-a_1(3))|_7 = |(5)(3)(3-1-4(3))|_7 = |(1)(-10)|_7 = 4$$
To find the residue relative to a new base modulus, 11, we must now compute
$$|1+4(3)+4(15)|_{11} = |1+1+4(4)|_{11} = |2+5|_{11} = 7$$
(Note that for this example n=73 which is indeed 7 mod 11.)

The second fundamental problem of base extension arises out of the time-penalty associated with base extension. This has components arising from reprogramming the RNS processors for new base moduli and the extension operation itself.

We consider first the operation count for base extension. There is a natural parallelism in the operations for extension of the members of a column of the matrix. In the above example, the computation of a_1 requires 2 arithmetic operations in the modulus-5 processor, followed by the computation of a_2 which can be viewed as three multiply-accumulate operations in the modulus-7 processor. Concurrent with this latter stage, the a_1 computation for the next element can be performed. The total time for the conversion to mixed radix form will therefore consist of the current vector length times the time for the final modulus calculation plus a latency comprised of the sum of the times for a single element to pass through the earlier channels. Let us examine this for an RNS basis of dimension L and active column length C.

The number of operations required in the i^{th} processor for the conversion of one entry to mixed radix is i except that the first processor is not needed. The total number of multiply-accumulate operations needed to obtain the full mixed radix representation of the first element is therefore $L(L+1)/2 - 1$. The rest of the column would require just a further $(C-1)L$ operations so that the total number of parallel operations needed is $L(2C+L-1)/2-1$.

This operation count must be increased since the elements of the pivot column must also be converted in each processor. This has the effect of doubling the vector length which gives a final count of $L(4C+L-1)/2-1$. For the first stage of our algorithm we have a vector length C=4 and an RNS basis of dimension L=4 so that the delay is 37 modular multiply-accumulate operation times. For the final elimination phase, C=2, L=8 and so the delay for this part of the conversion is 59 such operations.

For each new modulus a vector of effective length $2C$ must be processed. The operation consists of $L-1$ multiply-accumulate operations in this new modulus. For the same two stages as mentioned above this entails 24 and 28 operations respectively.

However before the last step here can be performed, the processors for the new modulus must be reprogrammed for this new modulus. The principal component of this operation is the loading of the appropriate look-up tables. These tables, for 8-bit moduli consist of two 256-byte tables each. Assuming an effective transfer rate of 4 bytes/clock cycle, this operation takes 128 cycles which is a greater time penalty that is entailed in the RNS-MRS conversion. We may assume that these operations are concurrent.

With a throughput of 1 multiply-accumulate/cycle, the overall delay caused by the base-extension is therefore around 150 cycles which is a large cost to absorb within our requirements for a high-speed, physically small unit. This is especially true since the dynamic range growth allowed for in the example and in the above calculations is far from worst case. It is because of this cost that some scaling is incorporated into our algorithm. However, inclusion of scaling has its own associated costs - both in time and accuracy. We begin by considering the cost of scaling in terms of the accuracy of the final solution.

4.4 Scaling and loss of precision

First suppose that at some stage of the elimination, we try to preserve the actual dynamic range by scaling and that the active matrix contains elements which are close to extremes of that range. As we commented earlier, each individual step can be viewed as an operation on a 2×2 complex matrix which we can write as $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

Suppose then that (the components of) each element of this matrix lie in the interval $[-M,M]$ and that we require a scaled version of the resulting element $ad-bc$ which also lies in this range. For complex arithmetic, the worst case implies that components of $ad-bc \in [-4M^2, 4M^2]$ which demands a scaling by a factor of the order of $4M$. This is comparable to scaling the elements of the matrix by $2\sqrt{M}$ in advance of the computation. It is *not* equivalent to this prescaling since at least some of the lower order part of the result is preserved but it can approach this effective loss of precision especially if we believe that we are working close to the limits of the dynamic range.

To see this consider the simplified situation of the multiplication of two 32-bit positive integers which are close to the limits of this range and the scaling of the result back to this same range. Denote the two factors by a and b and write $a = a_1 2^{16} + a_2,\ b = b_1 2^{16} + b_2$ where $a_1, a_2, b_1, b_2 < 2^{16}$. Now the scaling that is necessary is the replacement of the product by $[a*b/2^{32}]$ whereas the comparable prescaling would result in computing $[a/2^{16}]*[b/2^{16}] = a_1 * b_1$. For a_1, b_1 close to the extremes of the range $[a*b/2^{32}] < a_1*b_1 + \min(b_1,a_2) + \min(a_1,b_2)$ so that the relative difference is around $1/\max(a_1,b_1)$ which we are assuming to be only about 2^{-16}.

The point of this is that the effect of scaling, when the numbers are close to the extremes of the range, is similar to the effect of prescaling and this is roughly equivalent to *halving* the resolution of the representation.

Suppose that we tried to retain the same dynamic range throughout the computation. For our example, the components of the initial matrix occupied a dynamic range requiring about 19 bits and we conjectured using three 7-bit moduli so that we are indeed close to the extremes. Assuming that the growth is not quite as severe as the worst case, we might expect that the number of bits required would double at each stage. (This is not overly pessimistic since that rate of growth is indeed achieved in the final stage of our example.)

The scaling at each stage is, according to the preceding analysis, approximately equivalent to halving the stored precision of the matrix - at each stage. Three such halvings of the precision would be needed which is equivalent to having an initial matrix with fewer than three bits of precision. The immediate question is "What effect does such a degradation in the implied precision of the covariance matrix have on the solution?"

The standard error analysis for Gauss elimination is not immediately applicable since it assumes that the divisions are performed. However, we can obtain an answer to this question by performing some simple experiments.

Data from the ABF simulator was input to LabVIEW to test the adaptive beamforming algorithms. We conducted a few preliminary experiments on quantized data to see how the performance degraded with smaller and smaller resolution (# bits) in the data. The same data was used for each variation in resolution so that we could see the difference in the beam plots, for a given resolution.

There is a point in which quantization causes the Signal-to-Quantization Noise Ratio SNR_Q (dB) to become larger than the Signal-to-Thermal Noise Ratio, SNR_T. Given that there is 6dB $(10*\log 2^2)$ per bit increase in the SNR_Q, the number of bits, b, required for SNR_Q to equal the SNR_T is

$$b = \left\lceil \frac{SNR_T}{10 \log 2^2} \right\rceil$$

When b is chosen such that $SNR_Q > SNR_T$, the quantization noise acts like another jammer (but non-directional), whose power is greater than the desired signal. This would be one source of performance degradation.

The adapted beam patterns were computed for varying numbers of bits of precision in the input data and the absolute difference $|B(\theta_S) - B(\theta_J)|$ was compared to the desired solution. Here θ_S, θ_J represent the directions of the signal and the jammer respectively. In each case the jammer was J=30 dB noise so the maximum difference is about 30 dB due to the presence of thermal noise. The results of some experiments are presented in Table 1 below.

TABLE 1

Degradation of solution as input data resolution is eroded.
J = 30 dB Noise
Values of $|B(\theta_S) - B(\theta_J)|$

# bits	$\theta_J=30^\circ$			$\theta_J=45^\circ$	
	Trial 1	Trial 2	Trial 3	Trial 1	Trial 2
16	30 dB	30 dB	28 dB	32 dB	31 dB
5	28 dB	30 dB	29 dB	33 dB	32 dB
4	26 dB	39 dB	32 dB	30 dB	34 dB
3	28 dB	28 dB	31 dB	31 dB	31 dB
2	17 dB	20 dB	21 dB	22 dB	28 dB
1	9 dB	Singular matrix	17 dB	12 dB	15 dB

Initially this performance looks promising in that 3 or 4 bits resolution in the data appears to yield tolerably good accuracy in the solution. The actual beam patterns generated differed much more significantly from the "correct" solution but the results at the important bearings held fairly constant. However this first impression is somewhat misleading. For data vectors of length 16, data accuracy of 1 bit yields complex scalar products which require 6 bits for their storage. That is the final row of Table 1 is obtained using a matrix with approximately twice the resolution (in the sense of double the *wordlength*) of the effective resolution suggested by the above analysis if we do not allow the dynamic range to grow.

From Table 1, it appears that data resolution of 4 bits yields reasonable results. This is equivalent to an effective wordlength for the final solution of about 12 or 13 bits in the covariance matrix. Of course we have no strong evidence for the adequacy of this precision at this point - what we do have is evidence of the inadequacy of significantly less precision than this. Sensitivity of the solution to the precision in the weights was considered by Nitzberg [9].

The conclusion we draw is that some compromise between range extension (with its inherent costs in timing) and the use of scaling (with its cost in precision) is necessary in order to achieve acceptable (in both senses) results.

5. The elimination algorithm

In this section, we describe in some detail the algorithm proposed for the elimination phase of the solution, together with giving further consideration to the architectural requirements of the process. Throughout the discussion we shall consider just a four-antenna problem using $K=16$ data vectors so that the initial matrix elements consist of inner products of complex 16-vectors.

We have already seen that substantial element growth must be accommodated and that some scaling is necessary in order to keep this under control. In order to decide on how much growth can be allowed and what scaling is necessary, we give some consideration first to the dynamic range which could be achieved for the final stage of the elimination using 16 processors.

The Gaussian primes which can be stored in eight or fewer bits are 5, 13, 17, 29, 37, 41, 53, 61, 73, 89, 97, 101, 109, 113, 137, 149, 157, 173, 181, 193, 197, 229, 233, 241. The largest dynamic range which can be obtained using 16 of these is the product

$$73\times89\times97\times101\times109\times113\times137\times149\times157\times173\times181\times193\times197\times229\times233\times241 = 3.85\times10^{34}$$

which corresponds to about 115 bits. The effective range $[-1.92\times10^{34}, 1.92\times10^{34}]$ is sufficient for the Example 2 of the previous section but is not large enough to allow for worst case growth especially at some of the intermediate stages.

The elements of the initial covariance matrix, we suppose can be represented relative to the RNS-basis {73, 89, 97} which has a symmetric dynamic range of $M_1 = [-315104, 315104]$ which corresponds to slightly more than 19 bits of precision. Since both the real and imaginary parts consist of 2×16 products this allows the original data to be quantized to 7 bits which appears from Table 1 to offer sufficient accuracy to allow satisfactory tuning of the array.

The 16-processor array would in fact have each element of the initial matrix stored relative to the four-dimensional basis {73, 89, 97, 101} which already accommodates some of the growth for the next stage. For this next stage a five-dimensional basis can be used and the Gaussian prime 197 is added to the basis. Now, $73\times89\times97\times101\times197 = 12{,}539{,}268{,}473$ so that the available dynamic range at this stage is $M_2 = [-6{,}269{,}634{,}236,\ 6{,}269{,}634{,}236]$ which corresponds approximately to ±32.5 bits. Allowing for worst case growth from the original dynamic range would require a range of $[-4M_1^2, 4M_1^2] \sim [-3.97\times10^{11}, 3.97\times10^{11}]$ which is approximately 63.3 times that available. A scale factor $V \geq 63.3$ is therefore needed.

If the L-CRT is used for the factorization then V must be chosen so that M' is a power of 2. Of course to avoid the risk of overflow, the scaling must be done in advance of the computation. Now scaling the dynamic range at this stage to $\pm2^{32}$, that is choosing $M'=2^{32}$ is equivalent to scaling the elements of the matrix to a range of $\pm2^{15}$ and so choosing $V_1 = 315{,}104\times2^{-15} = 9.6162\ 10938$ will suffice.

For the next stage, we allow maximal growth of the dynamic range with an eight-dimensional basis by extending the RNS-basis to {73, 89, 97, 101, 197, 229, 233, 241}. This yields $M_3 = [-8.06\times10^{16}, 8.06\times10^{16}]$ or approximately $\pm2^{56.2}$ which must be compared with the demands of worst case growth from M_2 which, with the scaling already applied, results in the interval $[-2^{66}, 2^{66}]$ and so requires an effective scale factor around 2^{10}. Again the scaling must be done in advance of this stage of the computation. In order to attain a generated range $[-2^{56}, 2^{56}]$, it is necessary to scale the results of the previous stage to the range $[-2^{27}, 2^{27}]$ and so the scale factor required is $V_2 = 6{,}269{,}634{,}236\times2^{-27} = 46.7124\ 1519$. Note that this scaling carried out by the L-CRT requires the use of a 28-bit 2's complement adder.

Even the worst case growth at the final stage can be accommodated in the 16-dimensional RNS representation using the full basis above for which $M_4 = [-1.92\times10^{34}, 1.92\times10^{34}]$. (This is true since the final stage consists of "real × real - complex × complex conjugate" which has a smaller growth factor associated with it than the more general operations needed earlier.)

The scaling achieved here is essentially optimal. The final dynamic range available is approximately $\pm2^{113}$ and to keep within this the dynamic range used at the previous stage must be within $\pm2^{56}$ and in order to stay within this the largest "power of 2" dynamic range allowable for the previous step is $\pm2^{27}$. These are precisely the dynamic ranges achieved here.

Of course the effect of this scaling could be achieved by the initial quantization of the data. If we trace the scaling back to the original matrix then it is equivalent to the worst case of scaling the elements of the initial covariance matrix first to $\pm2^{15}$ and subsequently by a further 2 bits to $\pm2^{13}$. This in turn is equivalent to a quantization of the initial data to ±4 bits or, equivalently, 5 bits resolution.

Note that the scaling advocated here would result in a smaller loss of precision since the contributions of less significant bits are retained as long as possible. This, combined with the experiments reported earlier gives cause to expect this algorithm to yield satisfactory results.

Of course this prescaling is wanted at just the same stage of the computation as the base-extension and so the MRS-scaling algorithm should be considered as an alternative in order to take full advantage of this conversion.

We summarize the elimination phase as Algorithm 1 below. For simplicity in this description, we only refer to the first 16 RNS processors in the array and include only the operations concerned with the matrix itself and not those for the right-hand side vector. We also continue with the special case ofa 4×4 matrix.

ALGORITHM 1

The parallel RNS forward elimination algorithm

```
Input     4×4 matrix $A$ with its elements represented in RNS
     with basis {73, 89, 97, 101} but scaled so that
          $|a_{ij}| \le (73\times 89\times 97-1)/2 = 315104$

Initialize The 16 processors are initialized for the moduli
    {73,89,97,101; 73,89,97,101; 73,89,97,101; 73,89,97,101}
     We shall denote by $p_k$ the prime modulus in processor $k$.

1.   Scale the elements of $A$ using the L-CRT with
          $V = 9.616210938,\ M' = 2^{15}$
     using processors 4j-3 to 4j for column j.

2.   Processors 1-4:
          Reinitialize for mod 197
          Modulus vector is now:
    {197,197,197,197,73,89,97,101,73,89,97,101,73,89,97,101}
     Processors 5-16:
     Compute MRS representations of matrix elements:
          Processors 5-8       $a_{i1},\ a_{i2}$
          Processors 9-12      $a_{i1},\ a_{i3}$
          Processors 13-16     $a_{i1},\ a_{i4}$

3.   Compute $\langle a_{ij}\rangle_{197}$ in processors 1-4 using processor j for
     column j.

4.   For $i,j\ge 2$, compute
          $\langle a_{ij}^{(2)}\rangle_{p_k} = \langle a_{ij}a_{11} - a_{1j}a_{i1}\rangle_{p_k}$
     using
          Processors  2, 5-8 for j=2
          Processors  3, 9-12 for j=3
          Processors  4, 13-16 for j=4

5.   Scale $a_{ij}^{(2)}$ by L-CRT using processors as in Step 4 with
          $V = 46.7124\ 1519,\ M' = 2^{27}$

6.   Processors 3, 4, 9-16:
     Compute MRS representations of matrix elements:
          Processors 3, 9-12  $a_{i2}^{(2)},\ a_{i3}^{(2)}$
          Processors 4, 13-16 $a_{i2}^{(2)},\ a_{i4}^{(2)}$
     Processors 1, 2, 5-8:
          Reinitialize for moduli 229, 233, 241
```

Modulus vector is now:
{229,229,197,197,233,233,241,241,73,89,97,101,73,89,97,101}

7. Compute $\langle a_{ij}^{(2)} \rangle_{p_k}$ in processors 1, 2, 5-8 using
    Processors 1, 5, 7 for $a_{i2}^{(2)}$, $a_{i3}^{(2)}$
    Processors 2, 6, 8 for $a_{i2}^{(2)}$, $a_{i4}^{(2)}$

8. For $i,j \geq 3$, compute
$$\langle a_{ij}^{(3)} \rangle_{p_k} = \langle a_{ij}^{(2)} a_{22}^{(2)} - a_{2j}^{(2)} a_{i2}^{(2)} \rangle_{p_k}$$
using
    Processors 1, 3, 5, 7, 9-12 for $j=3$
    Processors 2, 4, 6, 8, 13-16 for $j=4$

9. <u>Processors 1, 3, 5, 7, 9-12</u>:
    Compute MRS representation of $a_{ij}^{(3)}$
<u>Processors 2, 4, 6, 8, 13-16</u>:
Reinitialize for moduli 109,113,137,149,157,173,181,193
Modulus vector is now:
{229,109,197,113,233,137,241,149,
73,89,97,101,157,173,181,193}

10. <u>Processors 2, 4, 6, 8, 13-16</u>:
    Compute $\langle a_{ij}^{(3)} \rangle_{p_k}$

11. <u>Processors 1-16</u>:
Compute
$$\langle a_{44}^{(4)} \rangle_{p_k} = \langle a_{44}^{(3)} a_{33}^{(3)} - a_{34}^{(3)} a_{43}^{(3)} \rangle_{p_k}$$

<u>Output</u>
$$\langle a_{ij}^{(i)} \rangle_{\vec{p}_k}, \quad j \geq i$$
where
$\vec{p}_1 = (73,89,97,101)$
$\vec{p}_2 = (73,89,97,101,197)$
$\vec{p}_3 = (73,89,97,101,197,229,233,241)$
$\vec{p}_4 = (73,89,97,101,109,113,137,149,157,173,181,193,197,229,233,241)$

In Algorithm 1 we have assumed the presence of a second group of sixteen processors which would be used for the corresponding operations on the right hand side. Of course if only one such group were available then the operations described here could be duplicated for the right hand side. Some economy would be possible. The duplication of the computation of MRS representations for the elements of the pivot row could be eliminated in favor of operating with the right-hand side vector on one set of processors. The effective vector length for the elimination steps themselves would however be increased by one by distributing the right-hand side across the processors.

In the event that 32 processors are available then further economies can be made to reduce the effective vector lengths by using some of the initially spare capacity. This is summarized in Figure 5 below.

FIGURE 5 Diagrammatic representation of Algorithm 1 - 32 processor version

Processor #	Base p_k	Store $\langle\cdot\rangle_{p_k}$ $i\geq1$	Scale	Compute MRS rep	Compute $\langle\cdot\rangle_{p_k}$	Compute $\langle\cdot\rangle_{p_k}$ $i\geq2$	Scale	MRS & new moduli	Compute $\langle\cdot\rangle_{p_k}$	Compute $\langle\cdot\rangle_{p_k}$ $i\geq3$	MRS & new mod, $\langle\cdot\rangle_{p_k}$	Compute $\langle\cdot\rangle_{p_k}$	Compute $\langle\cdot\rangle_{p_k}$
1	73	a_{i1}	a_{i1}	a_{i1}				229	$a_{i2}^{(2)}$		109 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
2	89	a_{i1}	a_{i1}	a_{i1}				229	$a_{i3}^{(2)}$	$a_{i3}^{(3)}$	109 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
3	97	a_{i1}	a_{i1}	a_{i1}				229	$a_{i4}^{(2)}$	$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
4	101	a_{i1}	a_{i1}	a_{i1}				229	$b_i^{(2)}$	$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$
5	73	a_{i2}	a_{i2}	a_{i2}		$a_{i2}^{(2)}$	$a_{i2}^{(2)}$	$a_{i2}^{(2)}$			113 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
6	89	a_{i2}	a_{i2}	a_{i2}		$a_{i2}^{(2)}$	$a_{i2}^{(2)}$	$a_{i2}^{(2)}$			113 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
7	97	a_{i2}	a_{i2}	a_{i2}		$a_{i2}^{(2)}$	$a_{i2}^{(2)}$	$a_{i2}^{(2)}$			137 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
8	101	a_{i2}	a_{i2}	a_{i2}		$a_{i2}^{(2)}$	$a_{i2}^{(2)}$	$a_{i2}^{(2)}$			137 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
9	73	a_{i3}	a_{i3}	a_{i3}		$a_{i3}^{(2)}$	$a_{i3}^{(2)}$	$a_{i3}^{(2)}$		$a_{i3}^{(3)}$	149 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
10	89	a_{i3}	a_{i3}	a_{i3}		$a_{i3}^{(2)}$	$a_{i3}^{(2)}$	$a_{i3}^{(2)}$		$a_{i3}^{(3)}$	149 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
11	97	a_{i3}	a_{i3}	a_{i3}		$a_{i3}^{(2)}$	$a_{i3}^{(2)}$	$a_{i3}^{(2)}$		$a_{i3}^{(3)}$	157 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
12	101	a_{i3}	a_{i3}	a_{i3}		$a_{i3}^{(2)}$	$a_{i3}^{(2)}$	$a_{i3}^{(2)}$		$a_{i3}^{(3)}$	157 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
13	73	a_{i4}	a_{i4}	a_{i4}		$a_{i4}^{(2)}$	$a_{i4}^{(2)}$	$a_{i4}^{(2)}$		$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
14	89	a_{i4}	a_{i4}	a_{i4}		$a_{i4}^{(2)}$	$a_{i4}^{(2)}$	$a_{i4}^{(2)}$		$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
15	97	a_{i4}	a_{i4}	a_{i4}		$a_{i4}^{(2)}$	$a_{i4}^{(2)}$	$a_{i4}^{(2)}$		$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
16	101	a_{i4}	a_{i4}	a_{i4}		$a_{i4}^{(2)}$	$a_{i4}^{(2)}$	$a_{i4}^{(2)}$		$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
17	73	b_i	b_i	b_i		$b_i^{(2)}$	$b_i^{(2)}$	$b_i^{(2)}$		$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$
18	89	b_i	b_i	b_i		$b_i^{(2)}$	$b_i^{(2)}$	$b_i^{(2)}$		$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$
19	97	b_i	b_i	b_i		$b_i^{(2)}$	$b_i^{(2)}$	$b_i^{(2)}$		$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$
20	101	b_i	b_i	b_i		$b_i^{(2)}$	$b_i^{(2)}$	$b_i^{(2)}$		$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$
21	197				b_i	$b_i^{(2)}$	$b_i^{(2)}$	$b_i^{(2)}$		$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$
22	197				a_{i2}	$a_{i2}^{(2)}$	$a_{i2}^{(2)}$	$a_{i2}^{(2)}$			173 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
23	197				a_{i3}	$a_{i3}^{(2)}$	$a_{i3}^{(2)}$	$a_{i3}^{(2)}$		$a_{i3}^{(3)}$	173 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
24	197				a_{i4}	$a_{i4}^{(2)}$	$a_{i4}^{(2)}$	$a_{i4}^{(2)}$		$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
25	197				a_{i1}			233	$a_{i2}^{(2)}$		181 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
26	233								$a_{i3}^{(2)}$	$a_{i3}^{(3)}$	181 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
27	233								$a_{i4}^{(2)}$	$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
28	233								$b_i^{(2)}$	$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$
29	241								$a_{i2}^{(2)}$		193 $a_{i3}^{(3)}$	$a_{i4}^{(3)}$	$a_{44}^{(4)}$
30	241								$a_{i3}^{(2)}$	$a_{i3}^{(3)}$	193 $a_{i3}^{(3)}$	$b_i^{(3)}$	$b_4^{(4)}$
31	241								$a_{i4}^{(2)}$	$a_{i4}^{(3)}$	$a_{i3}^{(3)}$ $a_{i4}^{(3)}$		$a_{44}^{(4)}$
32	241								$b_i^{(2)}$	$b_i^{(3)}$	$a_{i3}^{(3)}$ $b_i^{(3)}$		$b_4^{(4)}$

There are further economies that could be made here. The most obvious of which is that the initial basis could be augmented with 197 at no additional cost. This would eliminate the need for the first MRS conversion and base extension. This simplification was not incorporated into the diagram so that it adhered closely to Algorithm 1 as described above.

6. Back substitution

In this section, we are concerned with the back substitution phase of the solution process. Again the apparent need for divisions at each stage of the process is the source of potential difficulty. The method used to overcome this has the same consequent problems of element growth as for the forward elimination. We see that the postponement of the divisions can be rearranged so that they are eliminated entirely since only a scalar multiple of the weight vector is required.

At the beginning of this stage we have the upper triangular system given by the augmented matrix

$$\left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(4)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} & b_2^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} & b_3^{(3)} \\ 0 & 0 & 0 & a_{44}^{(4)} & b_4^{(4)} \end{array}\right]$$

We shall simplify the subsequent notation by rewriting this system as

$$U\vec{x} = \vec{c}$$

and denoting the elements of this matrix and right-hand side by u_{ij}, $c_i^{(1)}$.

We begin by describing the algorithm without paying any attention to the problems of element-growth and scaling that will be crucial to the realization of such an algorithm. The basic idea of the algorithm is to use a column-oriented (or *column-sweep*) algorithm with implicit multiplications on the left-hand side. The final row of the system represents the equation

$$u_{44}x_4 = c_4^{(1)}$$

and we substitute this into the previous equations to get

$$u_{44}\begin{bmatrix} u_{11} & u_{12} & u_{13} & 0 \\ 0 & u_{22} & u_{23} & 0 \\ 0 & 0 & u_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} u_{44}c_1^{(1)} - u_{14}c_4^{(1)} \\ u_{44}c_2^{(1)} - u_{24}c_4^{(1)} \\ u_{44}c_3^{(1)} - u_{34}c_4^{(1)} \\ c_4^{(1)} \end{bmatrix} = \begin{bmatrix} c_1^{(2)} \\ c_2^{(2)} \\ c_3^{(2)} \\ c_4^{(2)} \end{bmatrix}$$

The arithmetic on the right hand side must be performed but the multiplication on the left is not needed since we are only interested in the relative magnitudes of the weights.

At the next step, we can proceed similarly to get

$$u_{33}\,u_{44}\begin{bmatrix} u_{11} & u_{12} & 0 & 0 \\ 0 & u_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} u_{33}c_1^{(2)} - u_{13}c_3^{(2)} \\ u_{33}c_2^{(2)} - u_{23}c_3^{(2)} \\ c_3^{(2)} \\ u_{33}c_4^{(2)} \end{bmatrix} = \begin{bmatrix} c_1^{(3)} \\ c_2^{(3)} \\ c_3^{(3)} \\ c_4^{(3)} \end{bmatrix}$$

which in turn can be reduced to

$$u_{22}\,u_{33}\,u_{44}\begin{bmatrix} u_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} u_{22}c_1^{(3)} - u_{12}c_2^{(3)} \\ c_2^{(3)} \\ u_{22}c_3^{(3)} \\ u_{22}c_4^{(3)} \end{bmatrix} = \begin{bmatrix} c_1^{(4)} \\ c_2^{(4)} \\ c_3^{(4)} \\ c_4^{(4)} \end{bmatrix}$$

and finally

$$u_{11}\,u_{22}\,u_{33}\,u_{44}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} c_1^{(4)} \\ u_{11}c_2^{(4)} \\ u_{11}c_3^{(4)} \\ u_{11}c_4^{(4)} \end{bmatrix} = \begin{bmatrix} c_1^{(5)} \\ c_2^{(5)} \\ c_3^{(5)} \\ c_4^{(5)} \end{bmatrix}$$

We can now use the final right-hand side vector as the solution setting $x_i = c_i^{(5)}$ and therefore eliminate the division operation from the back substitution phase completely.

Clearly there is the possibility of substantial growth on the right-hand side of this system. At the beginning of the back substitution, the first row is scaled to $\pm 2^{15}$, the second to $\pm 2^{32}$, the third to $\pm 2^{56}$ while the fourth has a dynamic range of approximately $\pm 2^{113}$. The "cross-multiplication" operations of the process outlined above would generate growth up to a maximum of about 222 bits which could be accommodated by using the full array of processors if they can operate with 9-bit moduli. The potential growth of the elements of the right hand side is shown in Table 2. The additional 14 moduli 257, 269, 277, 281, 293, 313, 317, 337, 349, 353, 373, 389, 397, 401 would generate a sufficient dynamic range. However, the final RNS-to-binary conversion would require a binary wordlength of around 223 bits which is perhaps too large to be practical. (The final scalar product operation of the CRT could be performed with a conventional *floating-point* multiply-accumulate unit since the relative magnitudes of the weights not the exact values of the $c_i^{(5)}$ are required. Such a unit may be as efficient as a very long wordlength binary integer processor for this purpose.)

TABLE 2
Dynamic range required for element growth during back substitution

i	$u_{ij}, c_i^{(1)}$	$c_i^{(2)}$	$c_i^{(3)}$	$c_i^{(4)}$	$c_i^{(5)}$
1	$\pm 2^{15}$	$\pm 2^{130}$	$\pm 2^{188}$	$\pm 2^{222}$	$\pm 2^{222}$
2	$\pm 2^{32}$	$\pm 2^{147}$	$\pm 2^{205}$	$\pm 2^{205}$	$\pm 2^{222}$
3	$\pm 2^{56}$	$\pm 2^{171}$	$\pm 2^{171}$	$\pm 2^{205}$	$\pm 2^{222}$
4	$\pm 2^{113}$	$\pm 2^{113}$	$\pm 2^{171}$	$\pm 2^{205}$	$\pm 2^{222}$

Our algorithm has been based hitherto on the assumption of small (typically 8-bit) RNS processors which rules out any possibility of accommodating the full element growth and therefore necessitates the use of some scaling.

In the back substitution phase there is a choice to be made as to which factors in a product are to be scaled, and by how much. In the elimination phase factors from similar dynamic ranges were being multiplied but that is no longer the case. Initially, the later rows have greater dynamic ranges and therefore carry more significant bits than the earlier ones and so we choose to scale the fourth row. In order to use just the same 16 base moduli as before, the right-hand side vector must be kept within the dynamic range of $\pm 2^{113}$. It is necessary therefore for the first stage to scale the fourth row to $M' = 2^{55}$ before the arithmetic. This requires a scale factor $V = 5.3381\,9343 \times 10^{17}$. Similar scalings are performed at the subsequent stages and these are summarized in Table 3. However, since subsequent scalings are being applied only to the right-hand side, any scaling must be applied to all elements of this vector. The scale factors used for the subsequent stages are approximately 2^{34} and 2^{17}.

What is the effect of this scaling on the accuracy of the solution? Consider two quantities $A+a$, $B+b$ where we assume that A,B have similar magnitudes and so do a,b with a,b being much smaller than A,B. Then

$$\frac{A+a}{B+b} - \frac{A}{B} = \frac{aB-Ab}{B(B+b)} = O(a/B)$$

and so the error in estimating $(A+a)/(B+b)$ by A/B is of the order of (the reciprocal of) the scale factor. Thus the worst error in the relative weights which is introduced in this process is around 2^{-17} which is comparable with the accuracy of the initial covariance matrix.

TABLE 3

Dynamic range and scaling requirements for back substitution

i	$u_{ij}, c_i^{(1)}$	Scaled to	$c_i^{(2)}$	Scaled to	$c_i^{(3)}$	Scaled to	$c_i^{(4)}$	$c_i^{(5)}$
1	$\pm 2^{15}$		$\pm 2^{72}$	$\pm 2^{38}$	$\pm 2^{96}$	$\pm 2^{79}$	$\pm 2^{113}$	$\pm 2^{113}$
2	$\pm 2^{32}$		$\pm 2^{89}$	$\pm 2^{55}$	$\pm 2^{113}$	$\pm 2^{96}$	$\pm 2^{96}$	$\pm 2^{113}$
3	$\pm 2^{56}$		$\pm 2^{113}$	$\pm 2^{79}$	$\pm 2^{79}$	$\pm 2^{62}$	$\pm 2^{96}$	$\pm 2^{113}$
4	$\pm 2^{113}$	$\pm 2^{55}$	$\pm 2^{55}$	$\pm 2^{21}$	$\pm 2^{79}$	$\pm 2^{62}$	$\pm 2^{96}$	$\pm 2^{113}$

The algorithm consisting of the forward elimination and back substitution described here therefore provides a feasible solution to the adaptive beamforming problem using RNS arithmetic with an array of RNS processors to accommodate much of the element growth inherent in the integer solution of the system of linear equations.

7. Gauss-Jordan algorithm

The Gauss-Jordan algorithm in which we eliminate above the main diagonal as well as below offers a natural alternative for the problem and architecture under consideration. The argument against the Gauss-Jordan algorithm on conventional processors lies in the fact that the arithmetic demands of the algorithm are significantly greater than using Gauss elimination. On an array processor this ceases to be true; while on a vector processor the disadvantage is reduced by virtue of increasing the effective vector lengths in the later stages.

For our problem, the difficulties of the additional scaling requirements of the back substitution phase would be removed (or, at least, alleviated). The MRS conversions and base extensions would need to be performed for all rows of the matrix throughout the algorithm and therefore the scalings which are used in the forward elimination phase must also be applied to all rows. This penalty is reduced by observing that as the solution proceeds the number of zero elements is increasing and so the additional cost may be acceptable. The same scale factors that are used in the forward elimination would be suitable. As for conventional vector processors for much of the arithmetic the only additional cost is that entailed in increasing the vector lengths which could be quite small.

On the surface Gauss-Jordan offers an attractive alternative for this situation which will be considered further in subsequent work. The biggest difficulty may well result from the fact that the final diagonal will not be just a constant as in the back substitution here which may mean that the final division is unavoidable.

8. Conclusions

In this paper we have presented a possible algorithm-arithmetic-architecture combination for the solution of the adaptive beamforming problem. The solution uses RNS arithmetic and a modified Gauss elimination algorithm. The divisions inherent in this algorithm are eliminated at the expense of a small number of scaling operations and the adaptive use of an RNS processor array to accommodate some of the growth in the desired dynamic range implicit in the method.

The indications from *preliminary* simulation of the RNS processors suggest that this algorithm may provide significant speed-up over conventional processors running at similar clock speeds. However it is anticipated that the RNS processors will be capable of much greater clock speeds and therefore the potential speed-up may be substantial. Clearly such claims are subject to much more extensive and complete experiment and this is one of the immediate next tasks.

Other future investigations following on from this work will entail the use of alternative algorithms for the solution of the underlying problem as well as modifications to this algorithm in which, for example, the use of the core function and other scaling techniques will be investigated for the possibility of further improvements.

References

[1] A.Z. Baraniecka and G.A. Jullien, *Residue Number System Implementations of Number Theoretic Transforms in Complex Residue Rings*, IEEE Trans ASSP 28 (1980) 285-271.
[2] K. Gerlach, *Fast Orthogonalization Networks*, IEEE Trans Antennas and Prop, AP-34 (1986) 458-462.
[3] R.T. Gregory and D.W. Matula, *Base Conversion in Residue Number Systems* Third Symposium on Computer Arithmetic, IEEE, Washington DC, 1975, 117-125.
[4] M. Griffin, M. Sousa and F.J. Taylor, *Efficient scaling in the residue number system*, Proc IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, IEEE, New York, 1989
[5] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, 1986
[6] J.G. McWhirter, *Recursive Least-Squares Minimization Using a Systolic Array*, Proc. SPIE, 431, Real-Time Signal Processing VI, 1983, 105-112.
[7] J. Mellot, J. Smith, E. Strom and L. Smethwick, *The Gauss Machine: A GEQRNS DSP systolic array*, IEEE Trans Comp, to appear
[8] R.A. Monzingo and T.W. Miller, *Introduction to Adaptive Arrays*, Wiley, 1980.
[9] R. Nitzberg, *Computational precision requirements for optimal weights in adaptive processing*, IEEE Trans AES 16 (1980) 418-425.
[10] K.H. O'Keefe, *A note on Fast Base extension for Residue Number Systems with three moduli*, IEEE Trans Comp 24 (1975) 1132-1133.
[11] K.H. O'Keefe and J.L. Wright, *Remarks on Base Extension for Modular Arithmetic*, IEEE Trans Comp 22 (1973) 833-835.
[12] C. Rader and A. Steinhardt, *Hyperbolic Householder Transformations*, Linear Algebra

in Signals, Systems and Control, SIAM, Philadelphia, 1988, 186-208

[13] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, and F.J. Taylor, *Residue NNumber System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE, New York, 1986.

[14] N. Szabo and R. Tanaka, *Residue Arithmetic and its Application to Computer Technology*, McGraw-Hill, New York 1967

[15] C.F.T Tang, and K.J.R. Liu, *A Fully Parallel and Pipelined Systolic Array for MVDR Beamforming*, Report #TR 91-89, University of Maryland - College Park, 1991.

[16] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.

[17] S.M. Yuen, *Algorithmic, Architectural, and Beam Pattern Issues of Sidelobe Cancellation*, IEEE Trans. Aerospace and Electronic Systems, 25 (1989) 459-471.

DISTRIBUTION LIST
Report No. NAWCADWAR-94112-50

	No. of Copies
Office of Naval Research 800 N. Quincy St. Arlington, VA 22217	5
Defense Technical Information Center ATTN: DTIC-FDAB Cameron Station BG5 Alexandria, VA 22304-6145	2
U.S. Naval Academy Mathematics Department (ATTN: Peter R. Turner) Annapolis, MD 21402	10
Mission Avionics Technology Department Naval Air Warfare Center Aircraft Division Warminster P.O. Box 5152 Warminster, PA 18974-0591 10 copies for Code 5051; Barry J. Kirsch 2 Copies for Code 0471	12